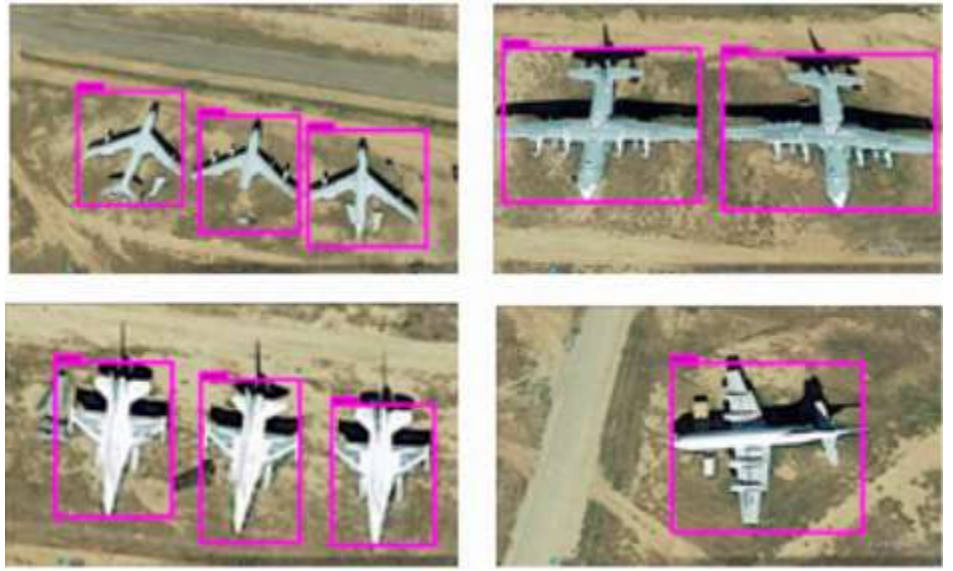
Neural Networks

ITI0210, lecture 12 (2021)

The Problem of Perception

Some problems are hard to solve by procedural programming

Example: computer vision



Radovic, Matija, Offei Adarkwa, and Qiaosong Wang. "Object recognition in aerial images using convolutional neural networks." *Journal of Imaging* 3.2 (2017): 21.

Difficult Functions

AlphaStar input in StarCraft II:

s_t - observations of over 10^5 variables during time $0 \ldots t$

Mapped to 10^{26} possible actions a_t by policy $\pi(a_t|s_t, z)$

The extremely difficult function $\pi()$

is computed by a neural network

Vinyals, Oriol, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature* 575.7782 (2019): 350-354.

Motivation

For problems that:

- are unreasonably difficult to compute
- or, we have no idea *how* to compute

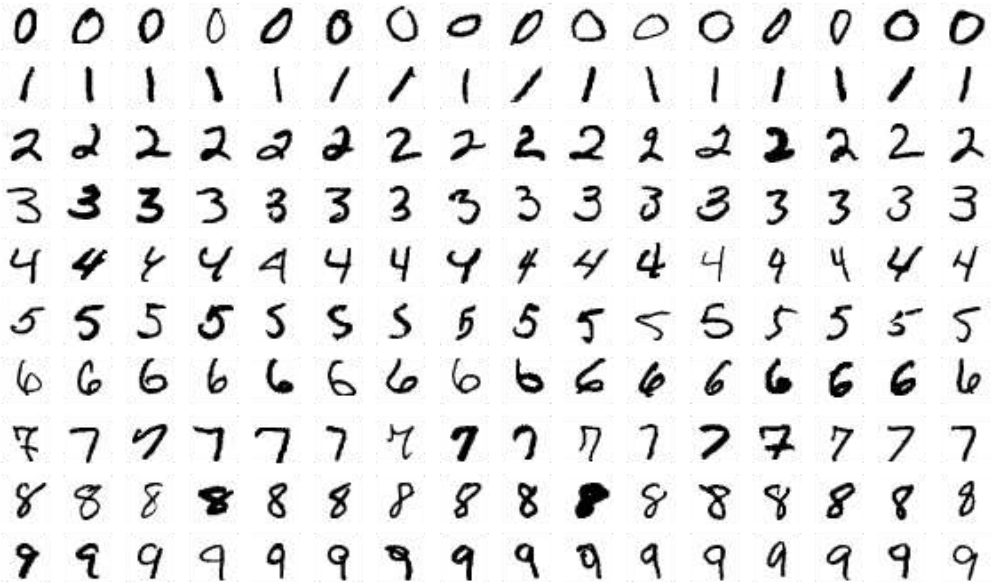
Machine learning may provide an **approximate** solution

Neural Networks

Usage Example

Problem: recognize handwritten digits automatically

You have some examples:



(Like the 60000 images

in the MNIST dataset;

https://en.wikipedia.org/wiki/MNIST_database)

Usage Example

Step 1: show the examples to the magic black box:

"These are 1-s"

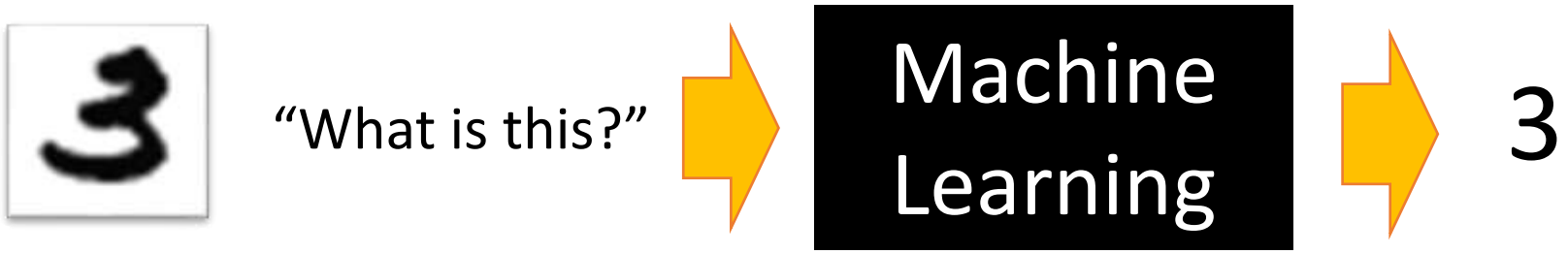
"These are 2-s"

"These are 3-s"

Machine Learning

Usage Example

Step 2: the magic box will start answering **based on the examples**



Problem solved!

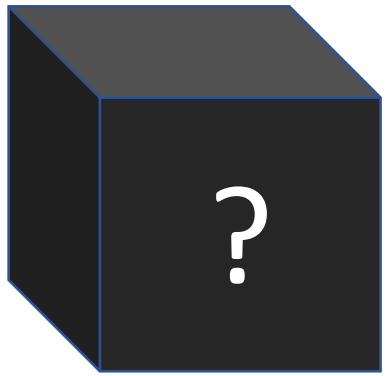
(Provided that it works reliably enough –

usually the data you needed this for should be similar to examples)

Making the magic box

There are many machine learning methods,
we will study only a few in this course

Starting focus: neural networks

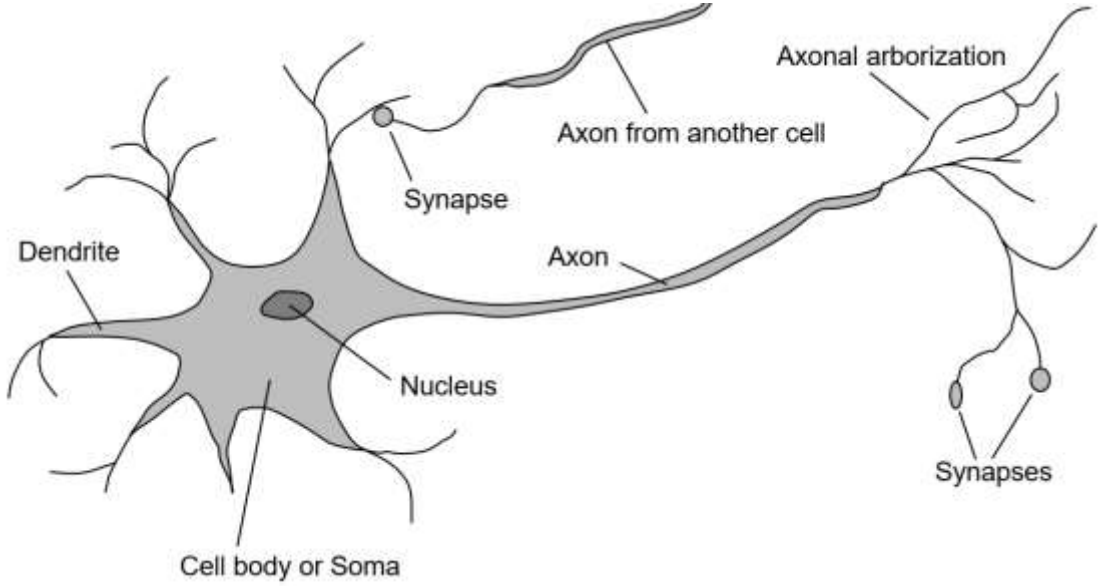


Artificial Neuron

Idea from 1950-s:

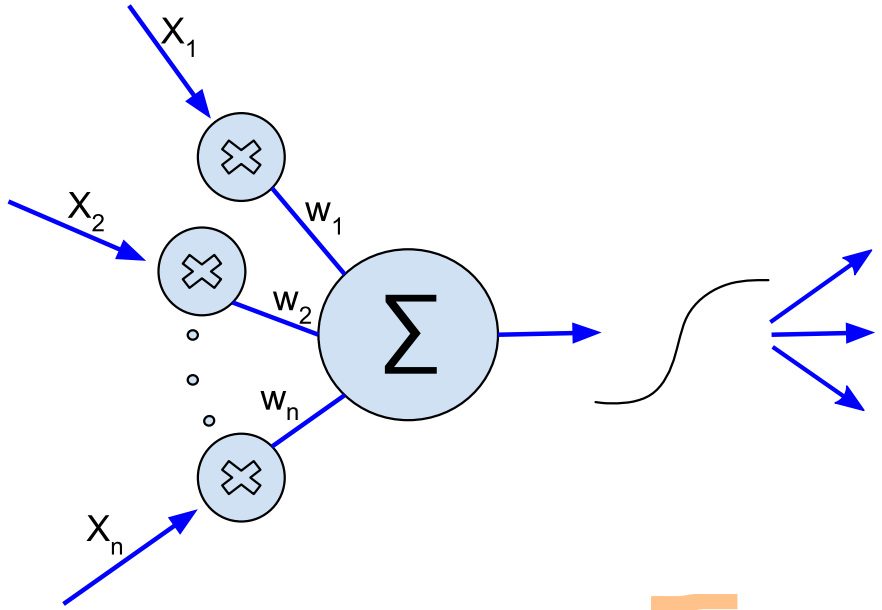
Can we imitate the brain to create AI?

consists of many similar units (neurons) that send signals to each other

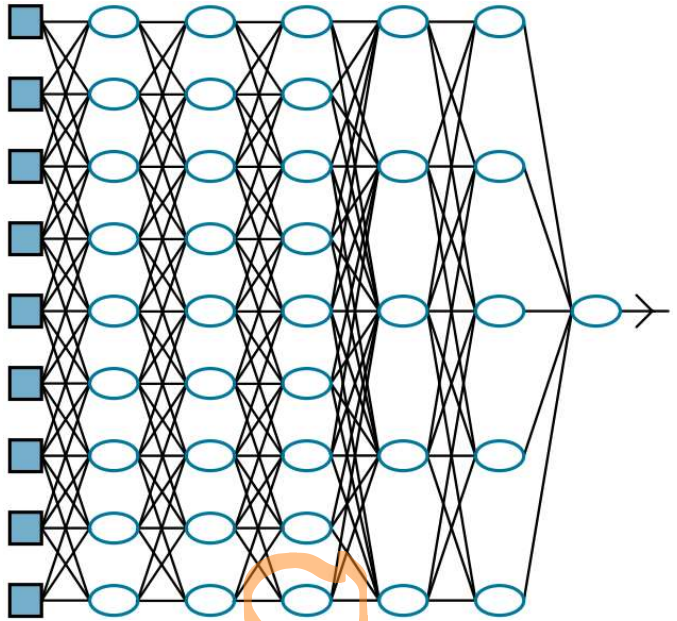


Artificial Neuron

Single unit:



Connect them like this:



Artificial Neuron

Inputs $x_1 \ldots x_n$, weights $w_1 \ldots w_n$

Bias weight w_0

Weights control which inputs we care about:

$$in = -w_0 + \sum_{i}^{n} w_i x_i$$

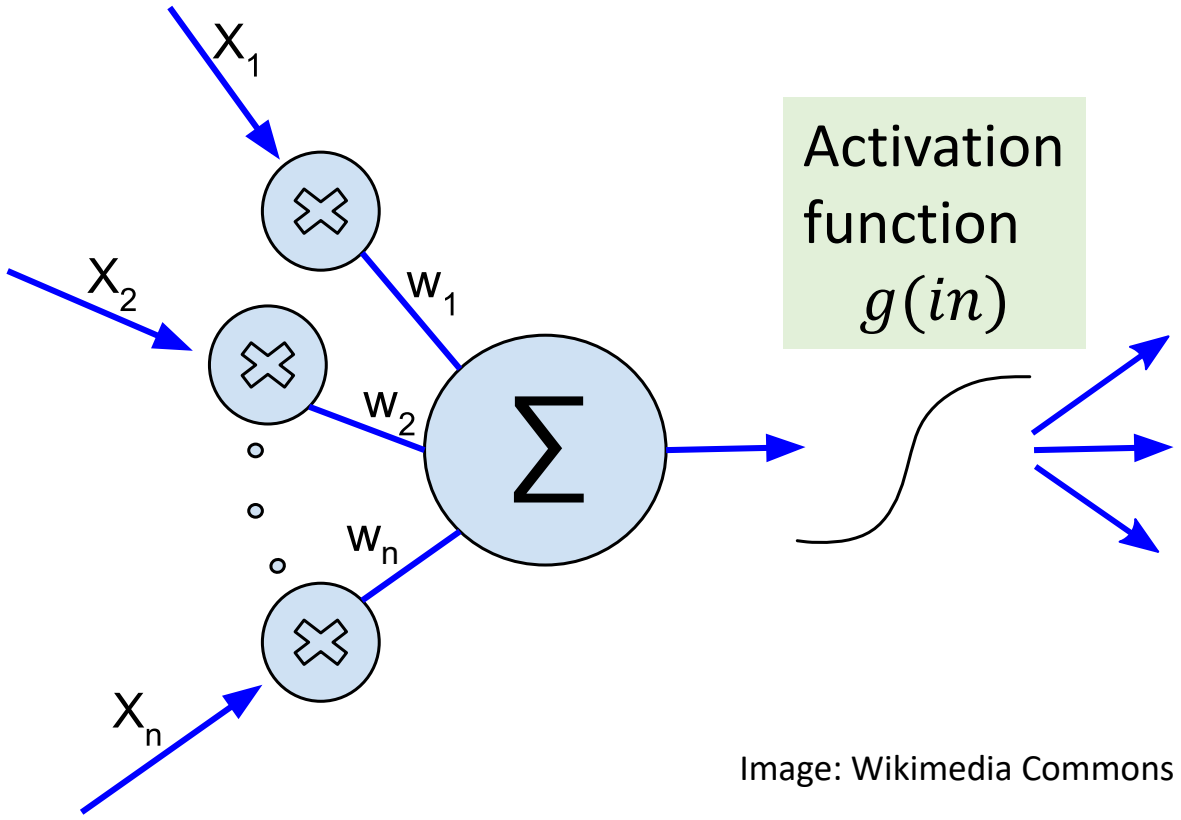


Image: Wikimedia Commons

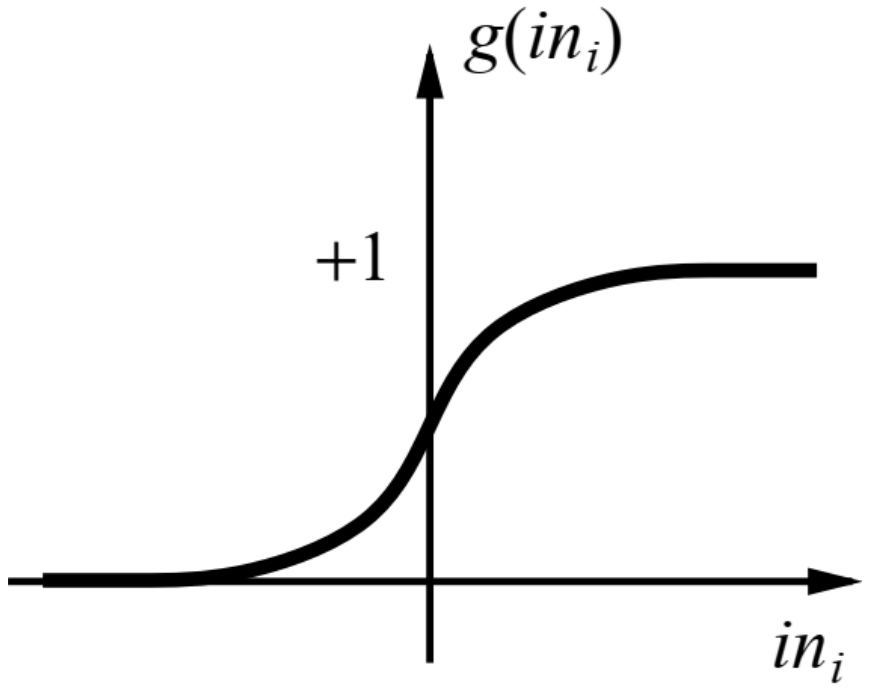
Artificial Neuron

Activation function controls the output signal of the neuron

Basic activation function is **logistic**

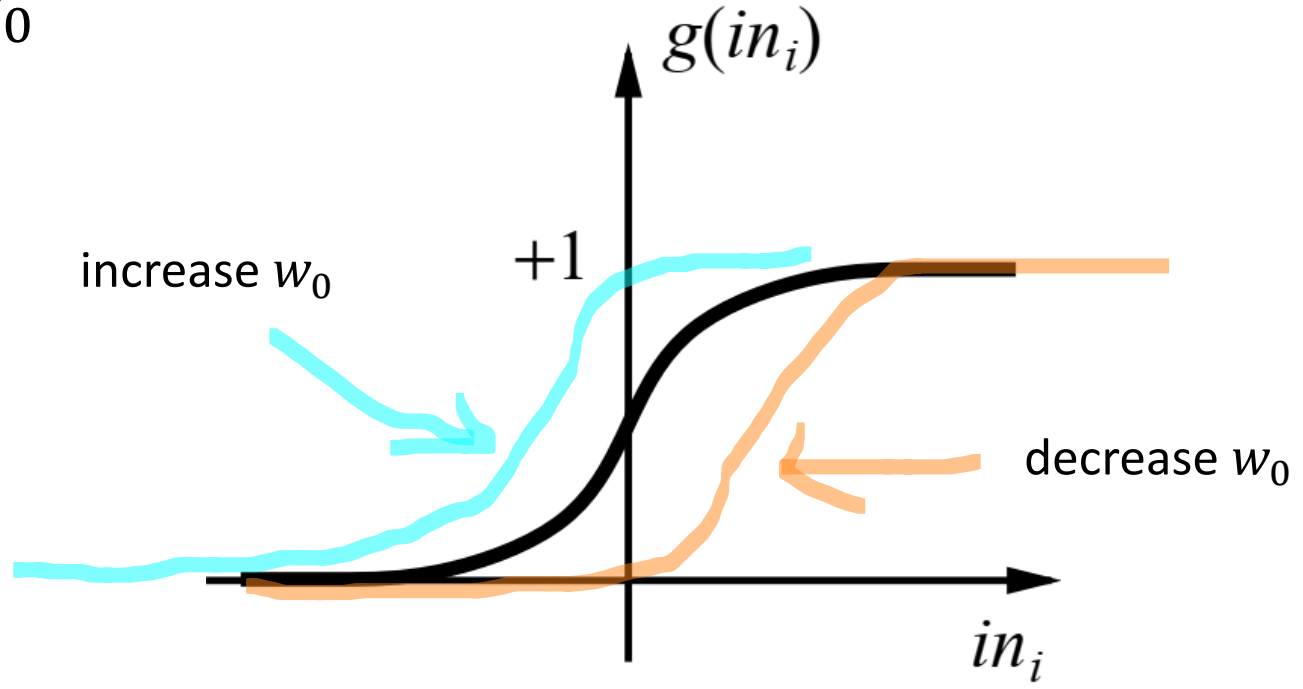
$$g(in) = \frac{1}{1 + e^{-x}}$$

Also called “sigmoid”



Artificial Neuron

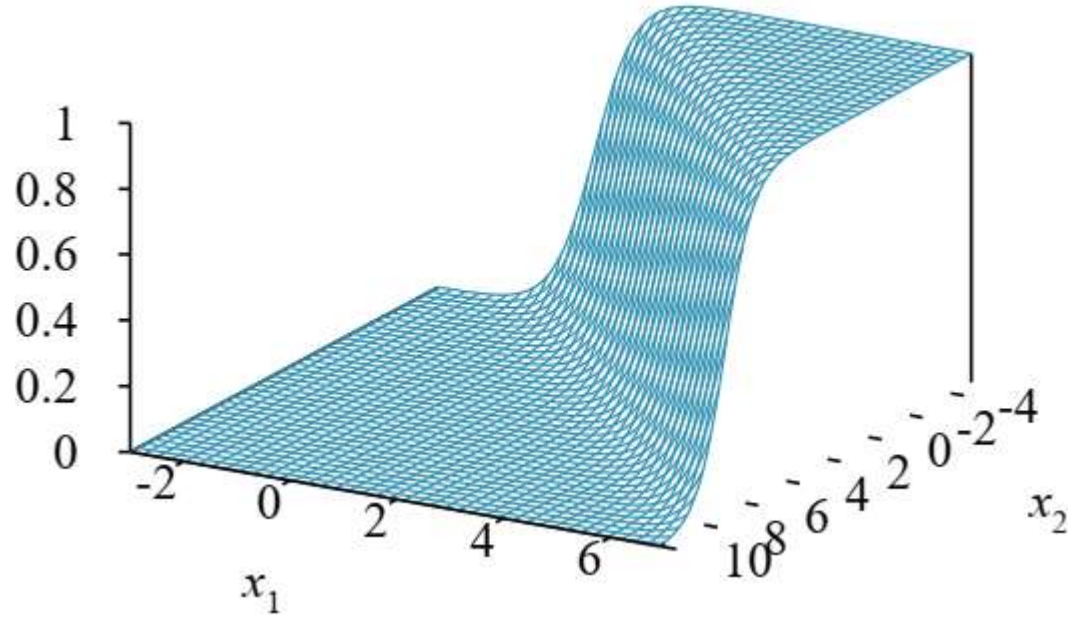
Role of bias weight w_0



Without bias, the curve is always positioned at 0

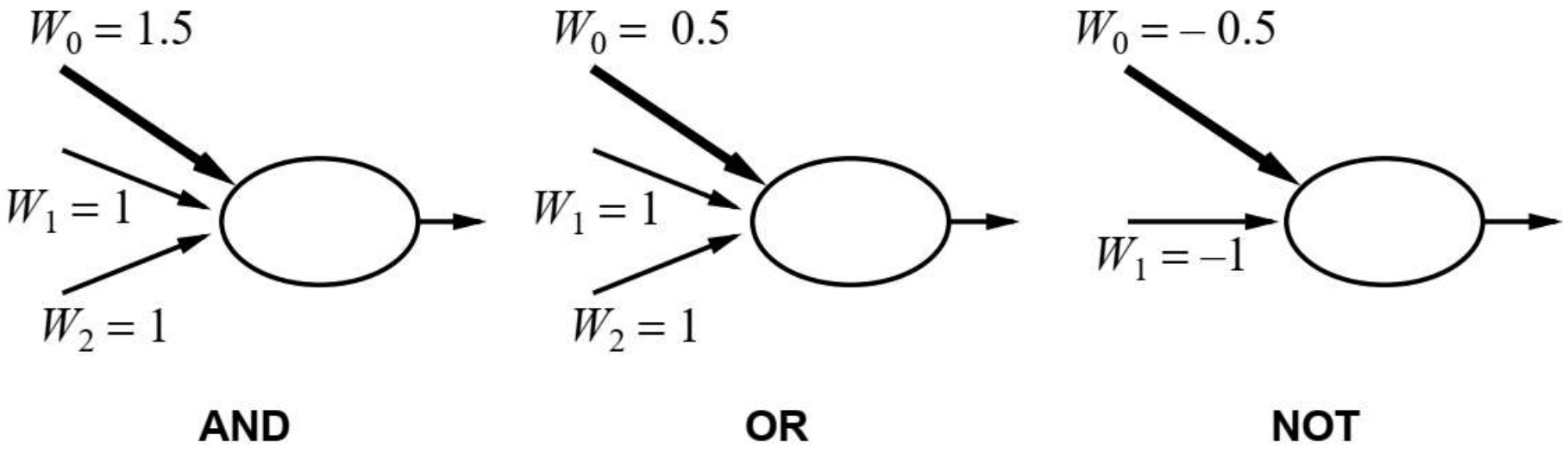
Artificial Neuron

Single neuron output with two inputs



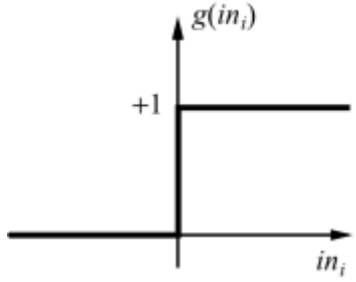
Understanding Single Neurons

Representing simple functions:



(Just an example – neural networks are not logic circuits)

Understanding Single Neurons



Use step function as activation here:

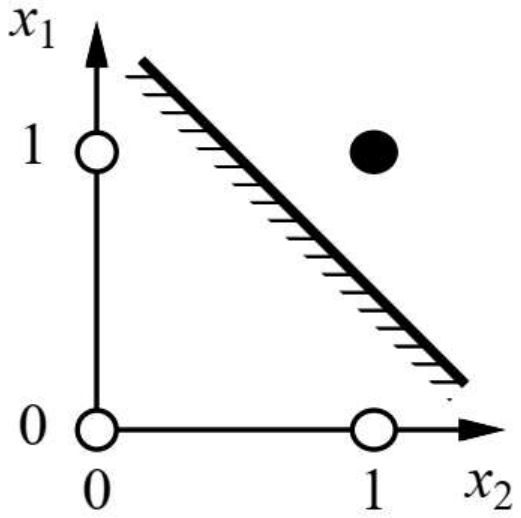
Logical AND

x_1	x_2	$x_1 \wedge x_2$	w_0	w_1	w_2	$in = -w_0 + w_1x_1 + w_2x_2$	$g(in)$
0	0	**0**	1.5	1	1	-1.5+1*0+1*0 = -1.5	**0**
0	1	**0**	1.5	1	1	-1.5+1*0+1*1 = -0.5	**0**
1	0	**0**	1.5	1	1	-1.5+1*1+1*0 = -0.5	**0**
1	1	**1**	1.5	1	1	-1.5+1*1+1*1 = 0.5	**1**

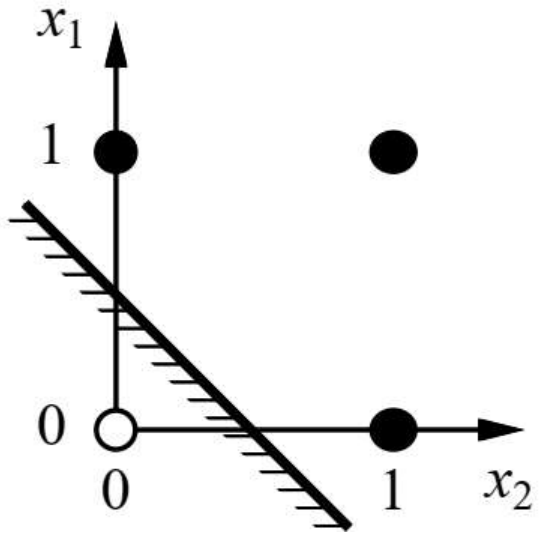
Understanding Single Neurons

Not everything can be represented by one neuron

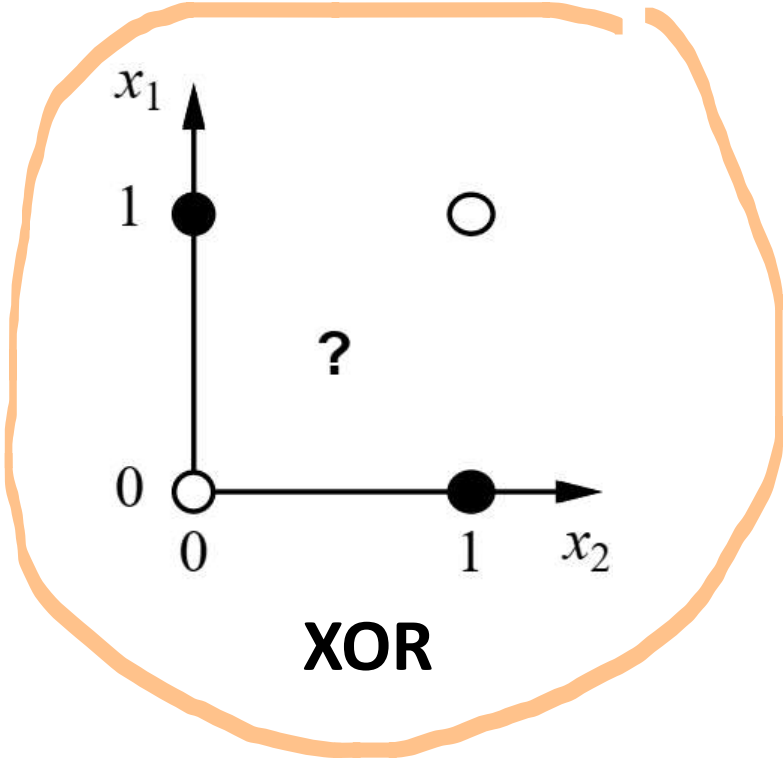
Challenge: draw a straight line to separate black and white dots



AND



OR



XOR

Let’s Try It Out

Single layer demo

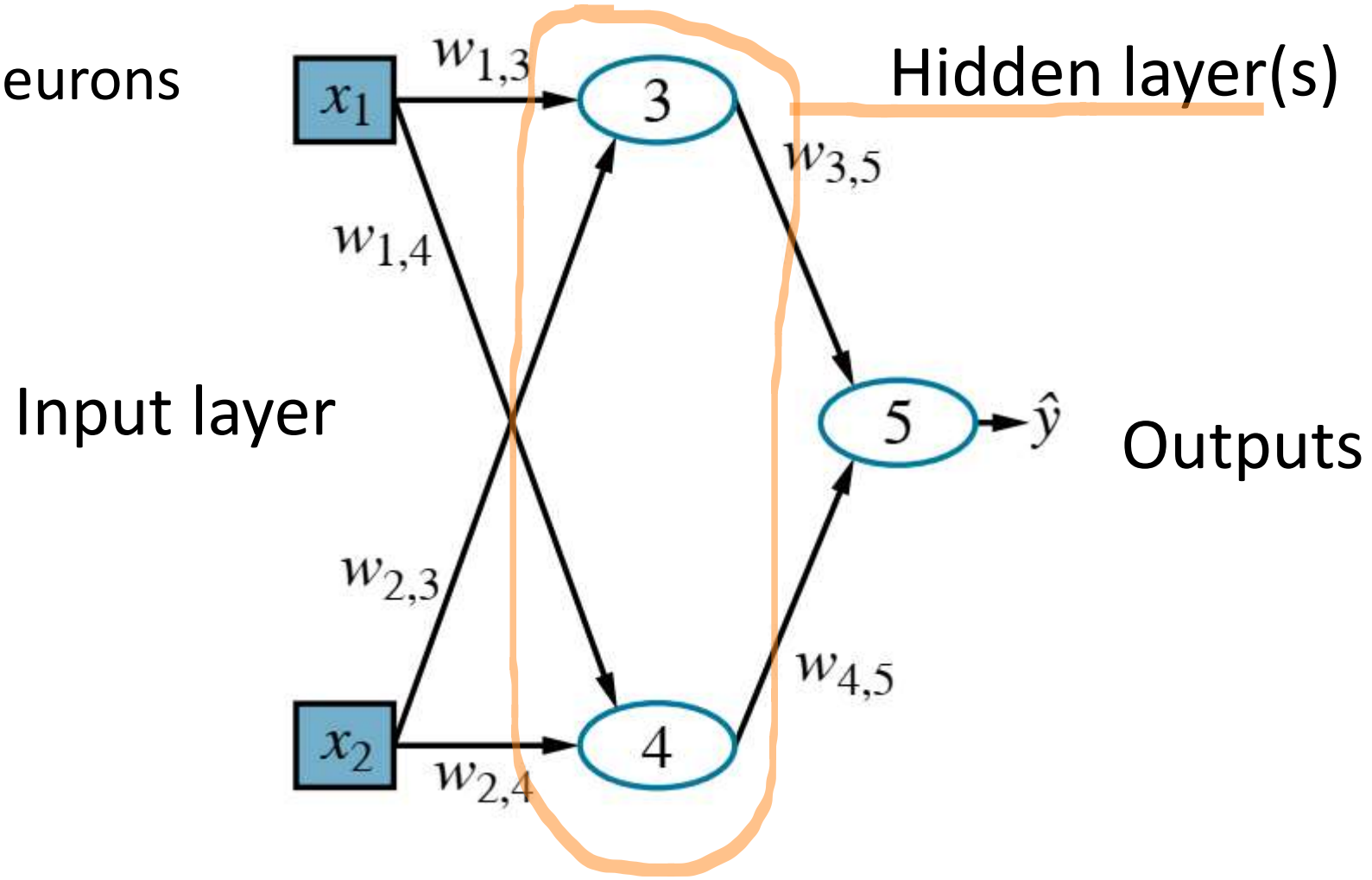
https://playground.tensorflow.org/

Adding Layers

Overcoming the limits of single neurons

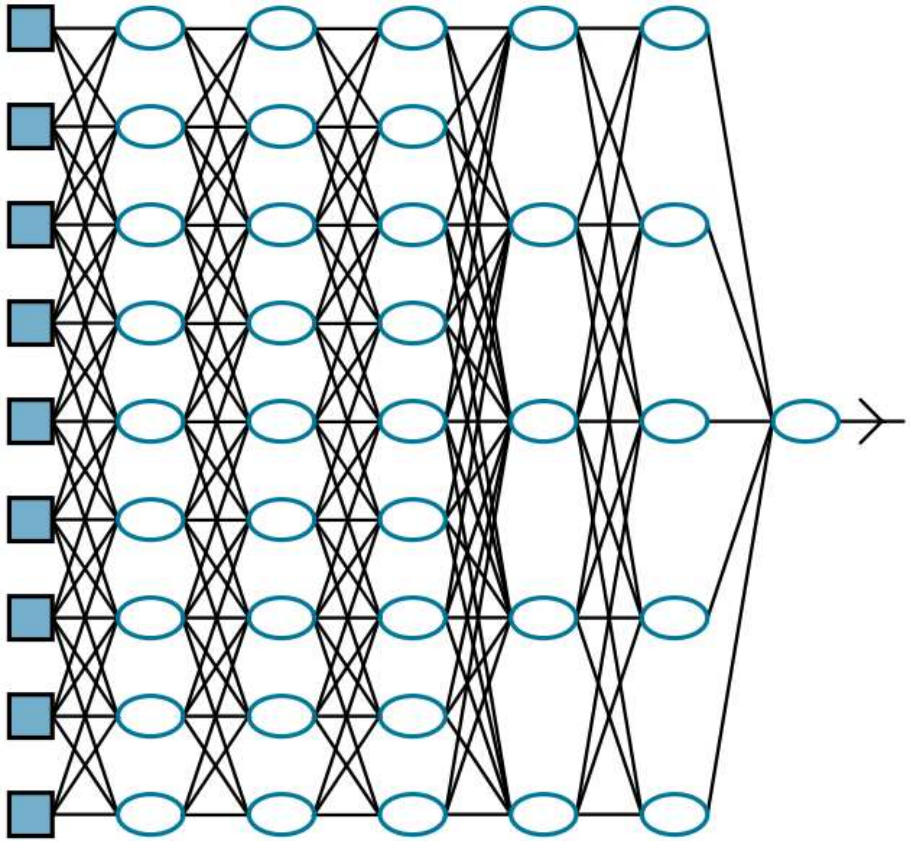
Multiple Layers

Network with 3 neurons



Multiple Layers

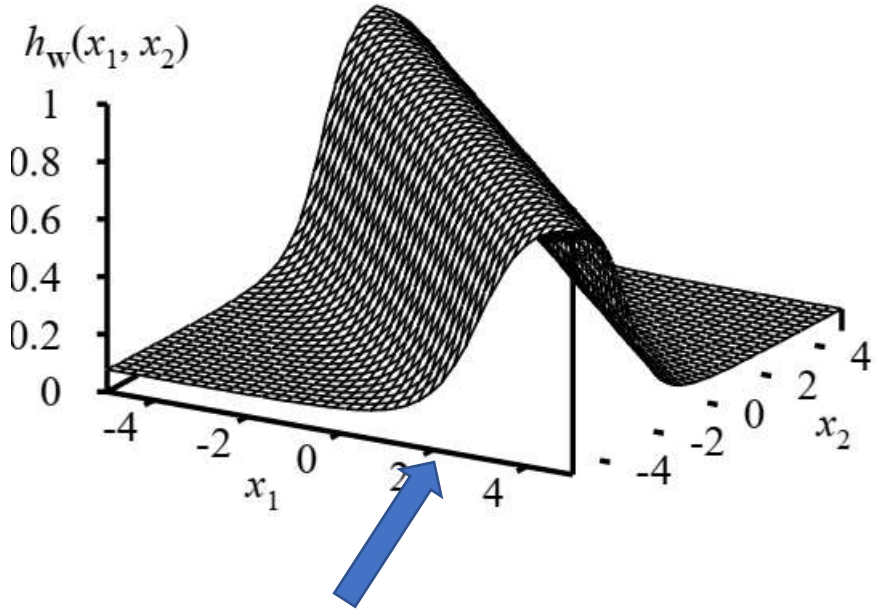
Deep network has
many hidden layers

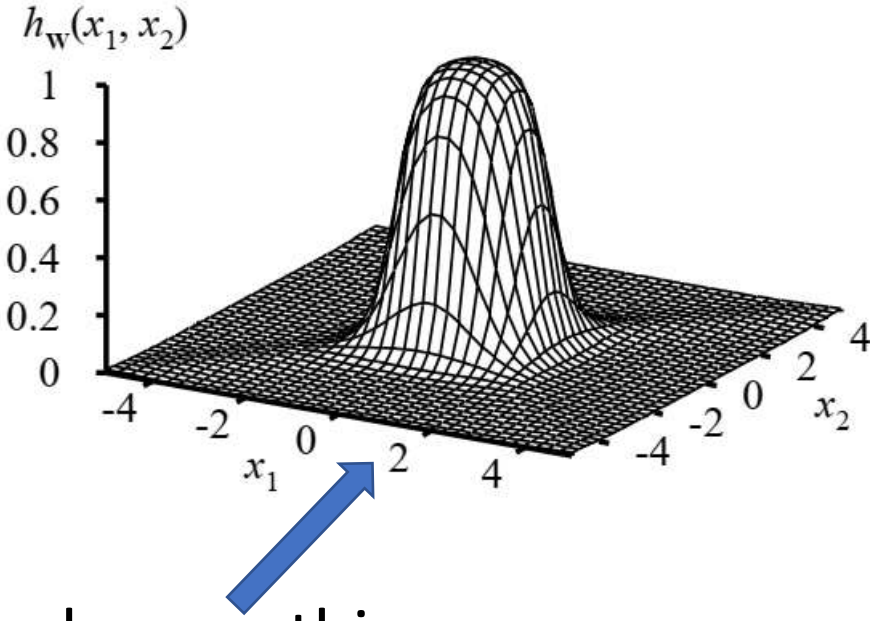


(ResNet-152 which you
can download has 151)

Effect of Hidden Layers

Single hidden layer already lets us learn these:





Here's the XOR we wanted and some other bumpy thing

Training Neural Networks

Idea level overview

Training a single neuron

We're classifying cats and dogs

$\hat{y}_k$: output for some neuron k

y_k : what we really wanted

$x_1 \ldots x_n$	$\hat{y}_k$	y_k	Error
Cat data	0.78	1	0.22
Dog data	0.33	0	-0.33
Cat data 2	0.96	1	0.04

Goal: try to make error close to 0

Training a single neuron

For neuron k

Output is $\hat{y}_k = g(-w_0 + \sum_i^n w_i x_i)$

For a single training example (one dog or cat), x_i and y are **fixed**

$$Error = y_k - g(-w_0 + \sum_i^n w_i x_i)$$

Training a single neuron

For neuron k

Output is $\hat{y}_k = g(-w_0 + \sum_i^n w_i x_i)$

For a single training

For one training example
Error is a function of the weights

y are **fixed**

$$Error = y_k - g(-\boldsymbol{w_0} + \sum_i^n \boldsymbol{w_i} x_i)$$

Weight Update

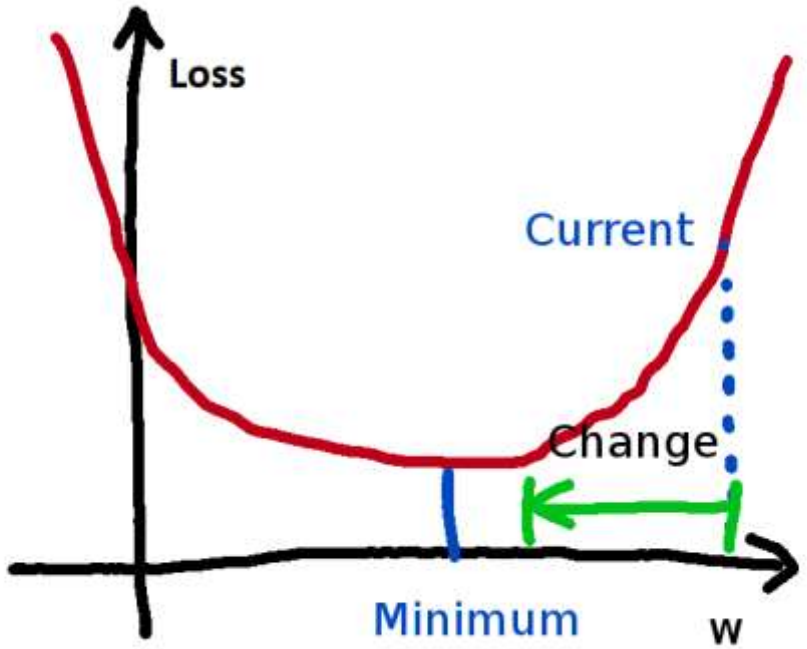
We can finally use that derivative we learned in high school!

$Loss = (y_k - \hat{y}_k)^2$ (Error squared)

Derivative $Loss'(w)$

gives direction of growth

Change w in opposite direction



Backpropagation

Where does y_k come from for hidden layers?

Actually need the error – send it back proportional to weights

(Instead of "error" we will start saying "gradient")